Einführung in die Informatik 1

– Datenorganisation und Datenstrukturen –

Sven Kosub

AG Algorithmik/Theorie komplexer Systeme
Universität Konstanz

E 202 | Sven.Kosub@uni-konstanz.de | Sprechstunde: Dienstag, 16:30-18:00 Uhr, o.n.V.

Wintersemester 2009/2010

Datenstrukturen

Datenstukturen

- Konzeptionen zur Organisation von Daten
- bauen auf elementaren Datentypen auf
- für die Effizienz von Algorithmen fundamental

Grundbausteine von Datenstrukturen

- **Reihung** (*array*)
  zur Verwaltung homogener Daten, d.h. für Daten, die vom gleichen Datentyp sind
- **Verbund** (*record*)
  zur Verwaltung heterogener Daten, d.h. für Daten, die nicht notwendig vom gleichen Typ sein müssen

Reihung

eindimensionale Reihung

- repäsentiert eine Folge von Daten gleichen Datentyps
- speichert Daten als Zeile einer Tabelle

- Repäsentierung von Funktionswerten an den Stellen i
- z.B. Messung der Signalstärke zu den Zeitpunkten $t = 1, \ldots, k$

Zeitpunkt	1	2	3	4	...	30	31
Signalstärke	10,5	10,5	12,2	9,8	...	13,1	13,3

- Zugriff auf jedes Element der Folge mit gleichem Zeitaufwand, z.B. in der Form `a[i]` auf das i-te Element
- sind die Daten vom Typ `T`, so ist die Reihung vom Typ `T`

Reihung

zweidimensionale Reihung

- repäsentiert eine Folge von eindimensionalen Reihungen vom Typ `T`
- speichert Daten in Matrixform, deren Zeilen Reihungen sind
- Zugriff durch `a[i,j]` auf das j-te Element der i-ten Zeile

- Temperaturverteilung über Gitterkoordinaten $t : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$
- Temperaturwert $t(1,1)$ an Stelle $(1,1)$ steht in Array `t` an Stelle `t[1,1]`

- alternativer Zugriff in Java: `a[i][j]`

n-dimensionale Reihungen als Verallgemeinerung

Reihung

Zeichenketten (*strings*)

- Reihungen vom Datentyp `char` (Buchstabe)
- typischerweise spezielle Syntax für Zeichenketten

Zeichenketten in Java:

- Verwendung von Anführungszeichen, z.B.

```
"Dies ist Text"
```

- zugehöriges Speicherbild:

D	i	e	s		i	s	t		T	e	x	t

- alternative Notation als (Standard-)Reihung:

```
{'D','i','e','s',' ','i','s','t',' ','T','e','x','t'}
```

Verbund

Verbund

- häufig Beziehungen zwischen Daten unterschiedlichen Typs zu modellieren

Stammdatenblatt `s`:

- `s.Name = "Mustermann"`
- `s.Vorname = "Martin"`
- `s.GebTag = 10`
- `s.GebMonat = 5`
- `s.GebJahr = 1930`
- `...`

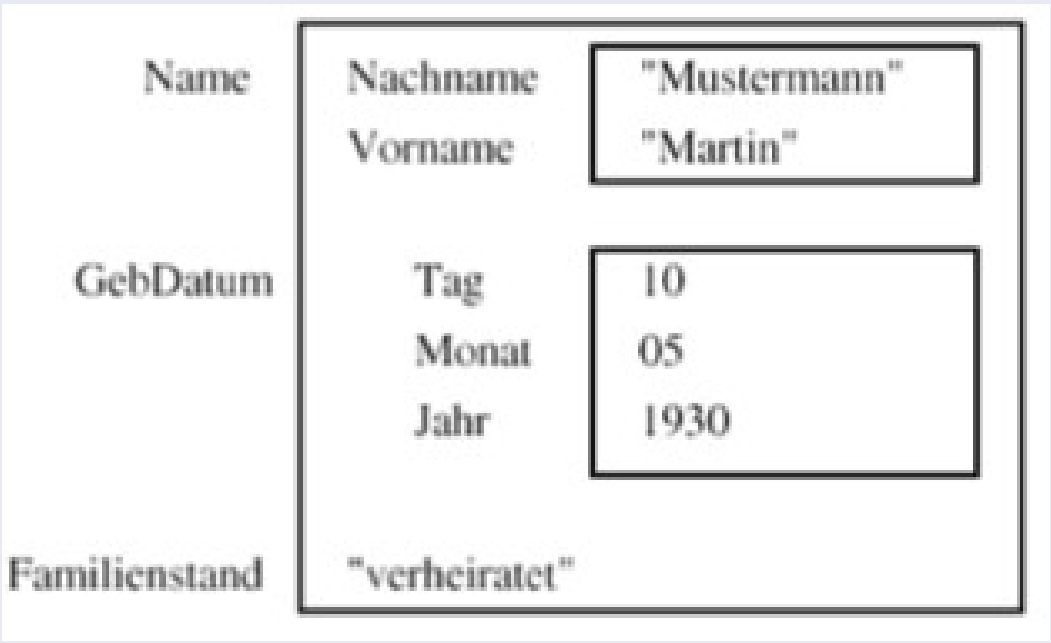
Name	"Mustermann"
Vorname	"Martin"
GebTag	10
GebMonat	05
GebJahr	1930
Familienstand	"verheiratet"
...	...

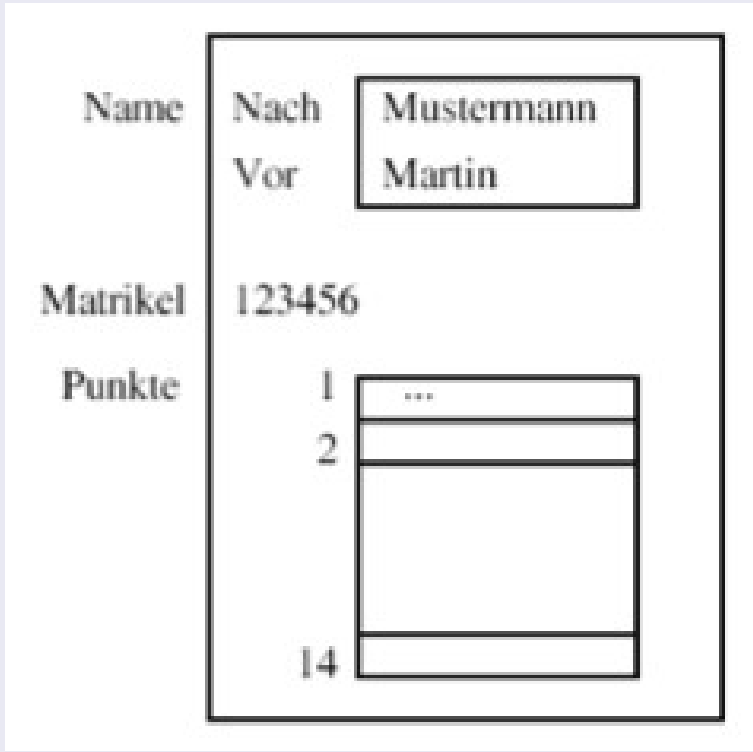
Verbund

Typkombinationen

- Komponenten eines Verbundes dürfen von beliebigem Typ sein
- insbesondere: vom Typ Verbund oder Reihung
- hierarchische Datenmodellierung

Komplexe Verbünde





Inklusionsbeziehungen

Wie ist ein Verbund in einem anderen Verbund enthalten?

- **Inklusion durch Wert** (*by value*)

  Daten des Unterverbundes werden im Verbund gespeichert

  \+ vollständige Speicherung an einem Ort
  − große Datenmengen pro Verbund zu speichern

- **Inklusion durch Referenz** (*by reference*)

  Daten des Unterverbundes werden an einem anderen Ort gespeichert und durch einen Verweis (Zeiger, *pointer*) auf die Speicheradresse gespeichert (referenziert)

  \+ flexibel; mehrere Zeiger auf gleiche Speicheradresse
  − (De)Referenzierung eine Hauptquelle für Fehler

Inklusionsbeziehungen

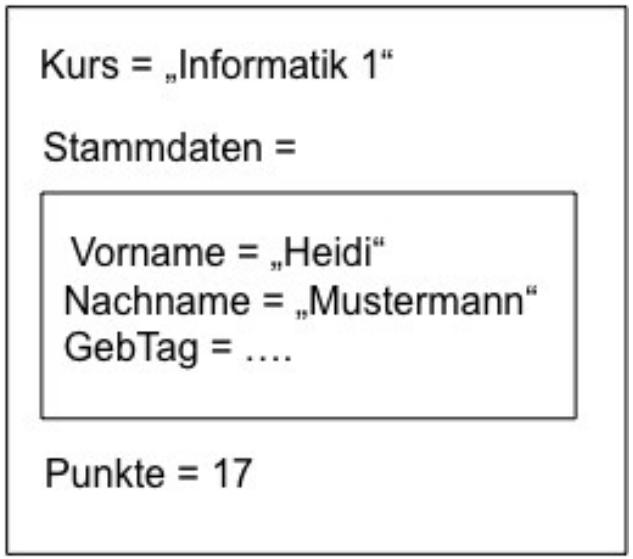
Beispiel-Szenario

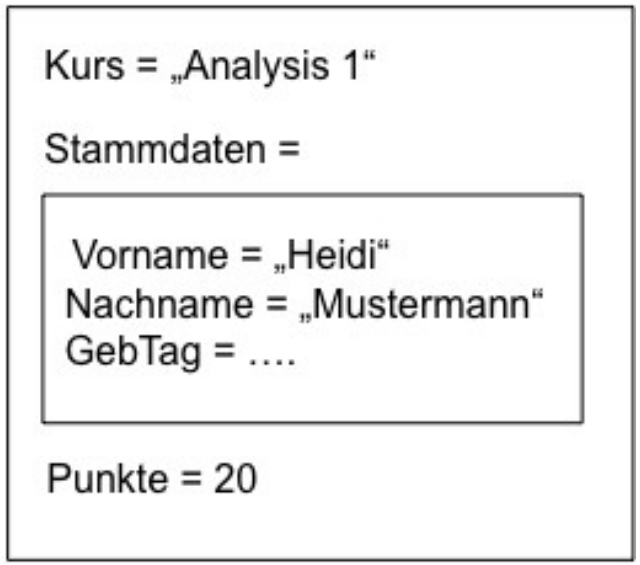
- Heidi Mustermann besucht Übungen zu zwei verschiedenen Vorlesungen
- jede Übung wird durch einen Verbund (Name, Punkte, etc.) realisiert
- d.h. Stammdaten für Heidi Mustermann in zwei Verbünden repräsentiert

Inklusionsbeziehungen

Inklusion durch Wert

- zwei separate Exemplare des Stammdatenblatts „Heidi Mustermann“
- doppelter Speicherplatz
- bei Änderung von Stammdaten müssen alle Exemplare gesucht und geändert werden

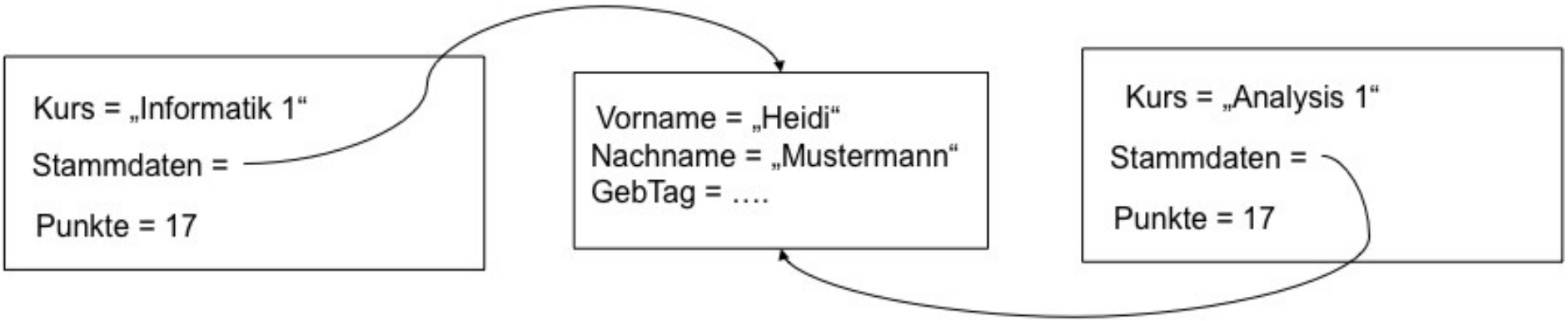




Inklusionsbeziehungen

Inklusion durch Referenz

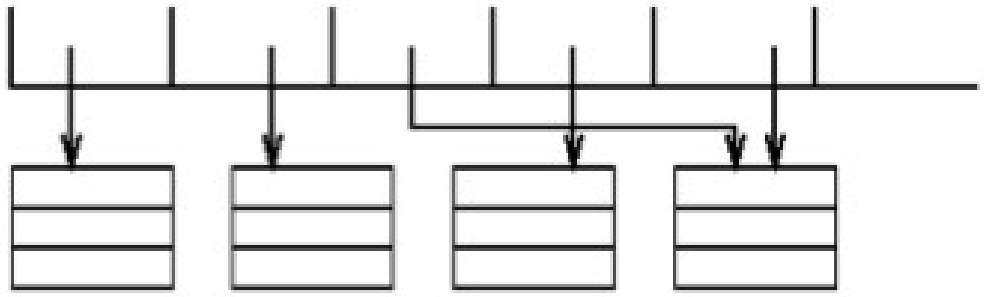
- ein Exemplar für Stammdatenblatt „Heidi Mustermann“
- beide Verbünde verweisen darauf
- zusätzliche Dereferenzierung notwendig



Inklusionbeziehungen

Inklusion durch Referenz (Forts.)

- Stammdatenblätter der Übungsteilnehmer durch Referenz
- Student hat sich zweimal angemeldet



Wie sollten Inklusionen modelliert werden - durch Wert oder Referenz?

- möglichst realitätsnah, d.h. im Beispiel: es gibt nur ein Exemplar einer Person, also nur ein Stammdatenblatt
- Methodik der objektorientierten Modellierung (demnächst)

Referenzen

Wie sollten Inklusionen modelliert werden - durch Wert oder Referenz?

- Effizienzüberlegung als weiterer Modellierungsaspekt
- wichtige Operationen auf Reihungen: Sortieren
- Typ `T` kann i.A. komplex sein
- im Schnitt $\approx n \cdot \log n$ Vertauschungen (bei n Elementen)
- Inklusion durch Wert: alle Daten müssen physikalisch vertauscht werden (i.A. viele Byte pro Vertauschung)
- Inklusion durch Referenz: nur Referenzen müssen vertauscht werden (4 Byte bei 32-Bit-Prozessoren)

Referenzen

Wie erfolgt Zugriff auf referenzierten Verbund?

C++

- Inklusion durch Wert mittels Punktnotation, z.B.

```
B[5].GebDatum.Tag
```

- Inklusion durch Referenz mittels Pfeilnotation, z.B.

```
B[5]->GebDatum.Tag
```

Java

- ausschließlich Inklusion durch Referenz möglich
- Punktnotation wird verwendet

Referenzen

Verbundhierarchien:

- Verbünde dürfen rekursiv geschachtelt sein, d.h. Verbund hält Referenz auf Verbünde mit den gleichen Komponenten
- typische Verwendung: Verwaltung dynamischer Datensammlungen, Navigation

Spezialfälle der rekursiven Referenzierung:

- **Listen**
- **Bäume**

Referenzen

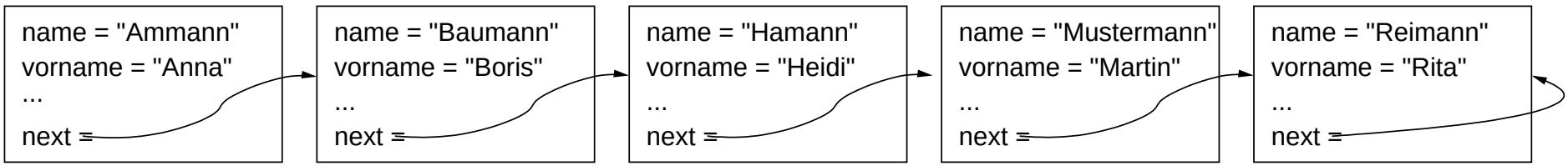
Liste:

- lineare Form der Datenorganisation
- erster Verbund der Liste heißt Kopf (`head`)
- jeder Verbund der Liste mit Referenz auf einen Nachfolgerverbund (`next`)
- jeder Verbund hat höchstens einen Verbund, der auf ihn referenziert
- letzter Verbund der Liste referenziert auf sich selbst oder auf speziellen Bezeichner für den leeren Verbund (`nil`)

Referenzen

Suchliste mit den studentischen Stammdaten:

- Verbünde der Liste enthalten Komponenten `name`, `vorname`, ..., `next`
- Verbünde in Liste aufsteigend lexikographisch nach `name` sortiert:
  - links von jedem Verbund sind nur Verbünde mit kleinerem `name` gespeichert
  - rechts von jedem Verbund sind nur Verbünde mit größerem `name` gespeichert



Referenzen

Algorithmus zum Suchen in Suchliste:

Algorithmus:	`listSearch(stud)`
Eingabe:	`stud` ist vom Verbundstyp der Liste
Ausgabe:	„Gefunden!“, falls Student mit Namen `stud.name` in Liste enthalten; „Nicht gefunden!“ sonst

```
(1) s=head;
(2) while (s≠nil und s.name≠stud.name) {
(3)   s=s.next;
(4) }
(5) if (s≠nil) { return(„Gefunden!“) }
(6) else { return(„Nicht gefunden!“) }
```

Referenzen

(binärer) Baum:

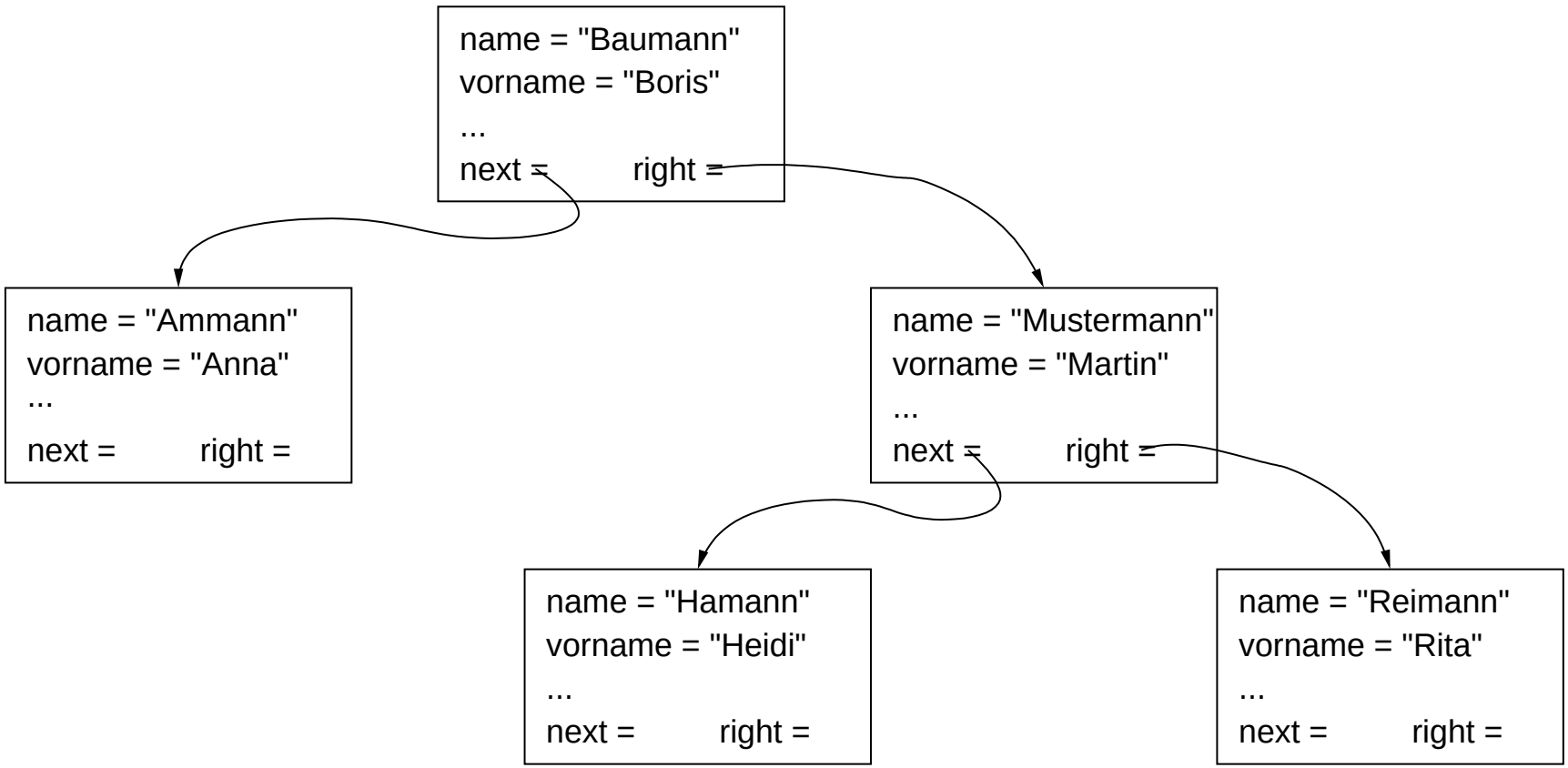
- zweidimensionale Form der Datenorganisation
- oberster Verbund des Baumes heißt Wurzel (`root`)
- jeder Verbund des Baumes mit Referenzen auf einen linken Kindverbund (`left`) und einen rechten Kindverbund (`right`)
- jeder Verbund des Baumes hat höchstens einen Verbund, der auf ihn referenziert
- unterste Verbünde des Baumes referenziert auf sich selbst oder auf speziellen Bezeichner für den leeren Verbund (`nil`)

Referenzen

Suchbaum mit den studentischen Stammdaten:

- Verbünde des Baumes enthalten Komponenten `name`, `vorname`, `...`, `left`, `right`
- Verbünde des Baumes wie folgt nach `name` sortiert:
  - links unterhalb jedes Verbundes sind nur Verbünde mit kleinerem `name` gespeichert,
  - rechts unterhalb jedes Verbundes sind nur Verbünde mit größerem `name` gespeichert

Referenzen


name = "Baumann"
vorname = "Boris"
...
next = right =
name = "Ammann"
vorname = "Anna"
...
next = right =
name = "Mustermann"
vorname = "Martin"
...
next = right =
name = "Hamann"
vorname = "Heidi"
...
next = right =
name = "Reimann"
vorname = "Rita"
...
next = right =

Referenzen

Algorithmus zum Suchen im Suchbaum:

Algorithmus: `treeSearch(stud)`
Eingabe: `stud` vom Typ `Student`
Ausgabe: „Gefunden!“, falls Student mit Namen `stud.name` in Baum enthalten, „Nicht gefunden!“ sonst

```
(1) s=root;
(2) while (s≠nil und s.name≠stud.name) {
(3)   if (s.name≤stud.name) { s=s.right; }
(4)   else { s=s.left; }
(5) }
(6) if (s≠nil) { return(„Gefunden!“) }
(7) else { return(„Nicht gefunden!“) }
```

Abstrakte Datentypen und Objekte

Abstrakte Datentypen (ADT):

- Einheit von Datenstrukturen und Manipulationsoperationen
- Zugriff nur über eine abstrakte Aufrufschnittstelle
- Interna bleiben verdeckt (*information hiding*)

Abstrakter Datentyp für Punkte in der Ebene (`Euclid2D`)

- interne Information:

  `x`, `y`, ...

- öffentliche Information:

  `getX()`, `getY()`, `set(x,y)`, `getAngle()`, `getRadius()`, `setPolar(angle,radius)`, `add(euclid2D)`, ...

- Java unterstützt ADT durch Zwang
- C/C++ überlässt Programmierern Verwendung von ADT